明确目标：

简单demo使用

能否“给缓存设置过期时间？”，“能否自动回收缓存中不需要的对象？”

如果能回收，回收的算法是什么？

导入maven依赖：

<dependency>

<groupId>com.github.ben-manes.caffeine</groupId>

<artifactId>caffeine</artifactId>

<!--https://mvnrepository.com/artifact/com.github.ben-manes.caffeine/caffeinez找最新版-->

<version>3.0.5</version>

</dependency>

创建实例过程：

Cache<Object, Object> cache = Caffeine.newBuilder()

//初始数量

.initialCapacity(10)

//最大条数🡪**大于这个值，会进行缓存淘汰**

.maximumSize(10)

//expireAfterWrite和expireAfterAccess同时存在时，以expireAfterWrite为准

//最后一次写操作后经过指定时间过期

.expireAfterWrite(1, TimeUnit.SECONDS)

//最后一次读或写操作后经过指定时间过期

.expireAfterAccess(1, TimeUnit.SECONDS)

//监听缓存被移除

.removalListener((key, val, removalCause) -> {事件监听器的内容，监听缓存移除})

//记录命中

.recordStats()

.build();

cache.put("1","张三");

//张三

System.out.println(cache.getIfPresent("1"));

//存储的是默认值

System.out.println(cache.get("2",o -> "默认值"));

如何保证线程安全：

既然是线程安全的“map”，那么自然就要考虑到“锁”🡪基于concurrenthashmap实现的，对数组中的头节点上锁以形成粒度小的分段锁，且扩容时不允许其他线程进行读写操作（甚至会让其他线程帮助扩容🡪将旧数据迁移到新concurrenthashmap中）

Hashmap的线程不安全例子：两次put，极端情况下会导致其中一个put操作丢失

淘汰机制：

Caffeine是基于**W-TinyLFU算法（目前只知道他对缓存进行分块了，用于处理有突发高并发的情况；并且用位图“粗略”地估算缓存中元素的使用频率，作为LFU淘汰的依据，好处是能节省内存；此外还会主动减少缓存中元素的使用频率，避免短期高频使用的元素长时间停留在缓存中）**进行缓存淘汰

目前确定有“惰性删除”，实际测试过，但定时删除没测试出来，姑且认为他没有

从缓存个数，权重（需要自己设定权重计算方式），按过期时间淘汰，引用（强引用，软引用，弱引用，虚引用🡪交给JVM进行删除）

用过期时间淘汰的问题是，假如我存储一个锁，之后很长一段时间都不会用上该锁，那它占用的内存空间就没用了🡪**但目前用户量并不大，就算所有用户的锁的key都存放进内存都是可容忍的**

淘汰内存是异步进行的

[解读JVM级别本地缓存Caffeine青出于蓝的要诀3 —— 讲透Caffeine的数据驱逐淘汰机制与用法 | HeapDump性能社区](https://heapdump.cn/article/5113544)

这篇文章讲用法的

[动手实现 LRU 算法，以及 Caffeine 和 Redis 中的缓存淘汰策略 - 风的姿态 - 博客园 (cnblogs.com)](https://www.cnblogs.com/fengzheng/p/13296082.html)

讲W-TinyLFU原理的🡪但没把我讲理解